`simulated_annealing` & scheduling

annealing	=	processo de aquecimento de metais, vidros, etc. seguido de resfriamento lento e gradual, com objetivo de tornar o material mais duro;
simulated	=	reproduzido por meio de um modelo, simulado;
scheduling	=	arranjo de eventos no tempo, escalonamento.

Como o próprio nome diz, o algoritmo *Simulated Annealing* explora uma analogia entre o modo como um metal se resfria e congela numa estrutura critalina de energia mínima (o processo real de *annealing*) e a busca por um mínimo num sistema qualquer.

À maneira de algoritmos de minimização de função como *Hill Climbing*, *Simulated Annealing*, em cada iteração, procura o próximo candidato a ponto de mínimo na vizinhança do candidato corrente, agindo de acordo com a diferença entre os valores da função-objetivo (chamada, nesse contexto, de função de *energia* ou *potencial*). A maior vantagem de *Simulated Annealing* sobre outros métodos, em especial *Hill Climbing*, é a possibilidade de evitar mínimos locais: o algoritmo emprega, para isso, uma busca aleatória que, por vezes, aceita vizinhos cuja energia seja mais elevada. Ou seja, em algumas iterações, o *Simulated Annealing* tende a maximizar a função-objetivo em vez de minimizá-la.

Entretanto, uma característica importante desse algoritmo é que a probabilidade de se aceitar um vizinho de maior energia decresce com o tempo, o que se implementa com um parâmetro, a temperatura, que decresce a cada iteração. Por fim, em qualquer temperatura, dados dois vizinhos de maior energia que o candidato a mínimo corrente, A e B, energia(A) > energia(B), a probabilidade de aceitação de A será menor que a de B.

O processo de minimização pode ser resumido no seguinte algoritmo:

```
candidato <- s₀;
T <- T₀;
repita
     próximo <- vizinho de 'candidato' tomado aleatoriamente[1];

     deltaE <- energia(próximo) - energia(candidato);
     se deltaE <= 0
        então candidato <- próximo
        senão faça candidato <- próximo com probabilidade
     exp(-deltaE/T)[2];
     T <- próximaTemperatura(T);
até T < Tfinal
retorna candidato;
```

onde:

- `s₀`: estado (candidato a mínimo) inicial;
- `T₀`/`Tfinal`: temperatura inicial/final;
- `próximaTemperatura`: função que calcula a temperatura vigente na próxima iteração;

Obs. (notas do algoritmo):

- **1:** um dos pontos fundamentais de diferença entre *Simulated Annealing* e *Hill Climbing*;
- **2:** expressão matemática que implementa a probabilidade variável de aceitação como explicada anteriormente.

Vale notar que `T₀`, `Tfinal` e `próximaTemperatura` são parâmetros de entrada adicionais do algoritmo, para os quais não há uma escolha única sempre eficaz.

`simulated_annealing, scheduling & Prolog (estudo-de-caso)`

Neste projeto, empregou-se *Simulated Annealing* para implementar uma solução para o problema de *scheduling*, instanciado como a criação de uma grade horária minimizando o número de períodos escolares, dados (i) um conjunto de disciplinas, (ii) seus respectivos pré-requisitos e (iii) o número máximo de disciplinas que podem ser cursadas por período.

Primeiramente, foi necessário proceder à modelagem do problema de forma a permitir a utilização de *Simulated Annealing*, o que inclui:

1. **reescrever o problema em termos de busca**, ou seja, definir um espaço de soluções. No presente projeto, cada ponto do espaço de soluções é, naturalmente, uma grade de horário completa e válida, embora não necessariamente ótima. De fato, pertencem ao espaço definido inclusive soluções "péssimas", com cada disciplina sendo cursada em um período exclusivo, o que se justifica pelo fato de não haver soluções melhores em certas configurações de dependência de pré-requisitos. O processo de busca implementado parte sempre de uma solução péssima, obtida pelo predicado `start/1`.

   Uma definição do espaço de soluções não estaria completa sem especificar sua "topologia", ou seja, quem é vizinho de quem. Definiu-se, portanto, que duas grades são vizinhas se e somente se (i) ambas são válidas, e (ii) uma pode ser obtida da outra por uma *transformação simples*. Uma transformação simples, por sua vez, consiste em "retirar" uma disciplina de uma grade e "reinseri-la" ou em outro período já existente ou em um novo período, "inserido" em algum ponto da seqüência original de períodos e correntemente contendo apenas a disciplina em questão. Eventualmente, a operação de "retirada" de uma disciplina deixará um período vazio, que é então suprimido.

   Vale notar que uma tal topologia tornaria uma busca a la *Hill Climbing* custosa demais, uma vez que a ordem de complexidade da escolha de melhor vizinho (operação intensamente

iterada) seria, nesse caso, exponencial, devido simplesmente ao excessivo número de vizinhos de um ponto qualquer. Isso, entretanto, não vale para *Simulated Annealing*, que jamais procura escolher o melhor vzinho, apenas um vizinho qualquer, operação que pode ser implementada de forma bastante eficiente, como exemplificado nesse projeto;

2. **determinar uma função de energia**, definida para todo ponto do espaço de soluções, cujo ponto de mínimo global coincida com uma solução ótima para o problema. No caso específico, definiu-se **energia(X)** como **o número de períodos da grade X**, uma escolha bastante óbvia;

A tarefa de implementação, realizada em Prolog, incluiu os seguintes passos:

1. **especificar uma estrutura de dados para grades de horário**, ou seja, especificar o formato dos pontos do espaço de soluções. Essa especificação foi realizada com vistas à eficiência da implementação do gerador de vizinhos aleatórios (próximo item) e compreende os seguintes pontos:

    - uma **grade** é uma estrutura de funtor **`sch`** e aridade **`3`** cujos argumentos são, respectivamente, **(i)** a **lista dos períodos** dessa grade, em ordem temporal inversa (1^o elemento = último período; 2^o elemento = penúltimo período; ...; último elemento = 1^o período), **(ii)** o **número total de períodos** dessa grade e **(iii)** o **número total de disciplinas** dessa grade;
    - um **período** é uma estrutura de funtor **`term`** e aridade **`2`** cujos argumentos são, respectivamente, **(i)** a **lista de disciplinas** desse período e **(ii)** o **número total de disciplinas** desse período;
    - uma **disciplina** é uma estrutura de funtor **`class`** e aridade **`2`** cujos argumentos são, respectivamente, **(i)** o **identificador** dessa disciplina e **(ii)** a **lista dos identificadores de seus pré-requisitos**, possivelmente vazia.

    Um exemplo de grade de horário seria o seguinte:

```
sch([
     term([
          class(vr,[ia,arq])
     ],1),
     term([
          class(ia,[filos,icc]),
          class(arq,[filos])
     ],2),
     term([
          class(filos,[]),
          class(icc,'[]')
     ],2)
],3,5)
```

2. **implementar um gerador de vizinhos aleatórios.** Em Prolog, uma abordagem mais natural à escolha de um vizinho qualquer teria sido definir um predicado `vizinho/2` que expressasse de forma lógica a relação de vizinhança e que, no *backtracking*, gerasse todos os possíveis vizinhos de uma dada solução. Dessa forma, no momento da escolha, bastaria encontrar a lista de todas os vizinhos (`findall`) e selecionar um elemento dessa lista de forma aleatória.

   É facil perceber, no entanto, quão ineficiente seria esse processo, motivo por que a equipe de desenvolvimento decidiu migrar para uma abordagem mais procedimental. Nesse sentido, implementou-se a escolha aleatória como a *geração aleatória de uma transformação simples a ser aplicada na solução atual*, resultando num vizinho aleatório, portanto. Uma transformação simples é "sorteada", pelo predicado `randomNeighb/3`, da seguinte forma:

   - escolhe-se, entre as disciplinas da grade, com igual probabilidade, uma disciplina a ser retirada (predicado `randomClassOut/3`);
   - escolhe-se, com igual probabilidade, entre (i) "inaugurar" um novo período com a disciplina escolhida e (ii) inseri-la num período já existente (predicados `randomClassIn/3`, `randomInsOp/3`);
   - escolhe-se, com igual probabilidade, entre os possíveis pontos de inserção do novo período/disciplina (idem).

3. **implementar o algoritmo de *Simulated Annealing*** (predicado `anneal/4`), em expressão geral, tendo ainda em aberto a função de resfriamento de temperatura (predicado `nextT/5`, cuja definição, na verdade, cabe ao usuário do programa);

4. **implementar um gerador de solução inicial, péssima** (predicado `start/1`).

executando_o programa

O programa implementado, `annealing`, foi desenvolvido com o pacote Amzi! Prolog + Logic Server 4.1, mas deve ser executado sem problemas em qualquer outro ambiente Prolog disponível.

O uso desse programa para a resolução de problemas reais não é muito simples, visto envolver a obtenção de alguns parâmetros não-triviais, como as temperaturas inicial e final e até mesmo a função de resfriamento da temperatura. Por isso, incluem-se na listagem do programa instruções mais detalhadas de uso e um exemplo bem simples de configuração de entrada.